LISTA SIMPLE

Una lista es un tipo de estructura de datos que forma una cadena mediante un enlace entre un dato y el siguiente.

El límite máximo lo impone la memoria, debido a que cada conjunto de datos debe ubicarse en una posición de memoria distinta, esta debe ser solicitada mediante el uso de las funciones de asignación dinámica.

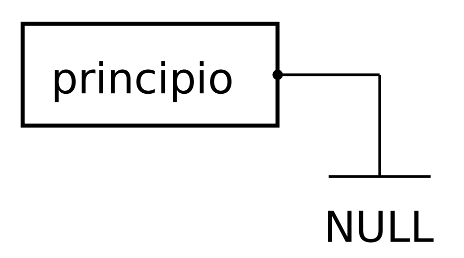
Para poder utilizar este tipo de estructura se deben agrupar los datos con un puntero

Se usa una estructura y dentro de ella un puntero a la misma

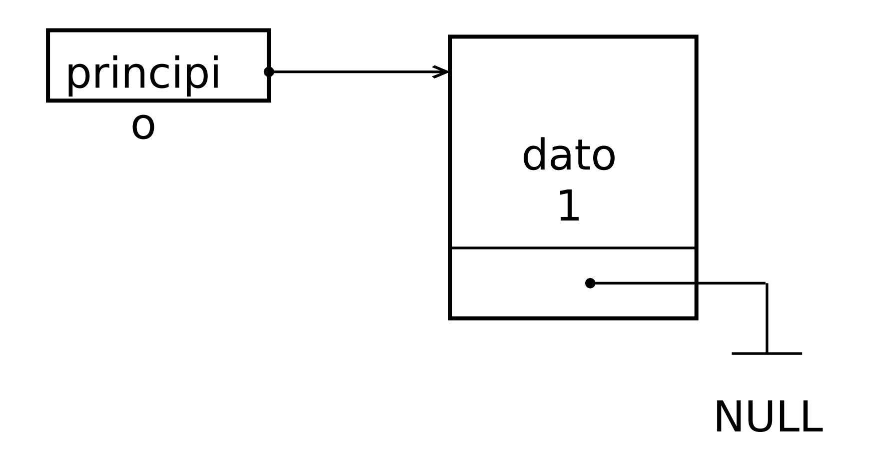
Este tipo de estructura se denomina estructura autoreferenciada.

Debemos un indicador al principio de los datos, esto lo debemos realizar mediante un puntero a estructura.

El puntero al comienzo, debe apuntar a un valor NULL, que indica que la lista esta vacía.



Cuando se genera el primer conjunto de datos el puntero de comienzo debe apuntar a este conjunto y el puntero del dato deberá apuntar a NULL



La ubicación de los datos puede hacerse ordenada según una determinada llave.

Este método me permite insertar cada conjunto de datos en el lugar adecuado según el orden deseado.

Se pueden presentar tres posibilidades

nsertar como primer elemento

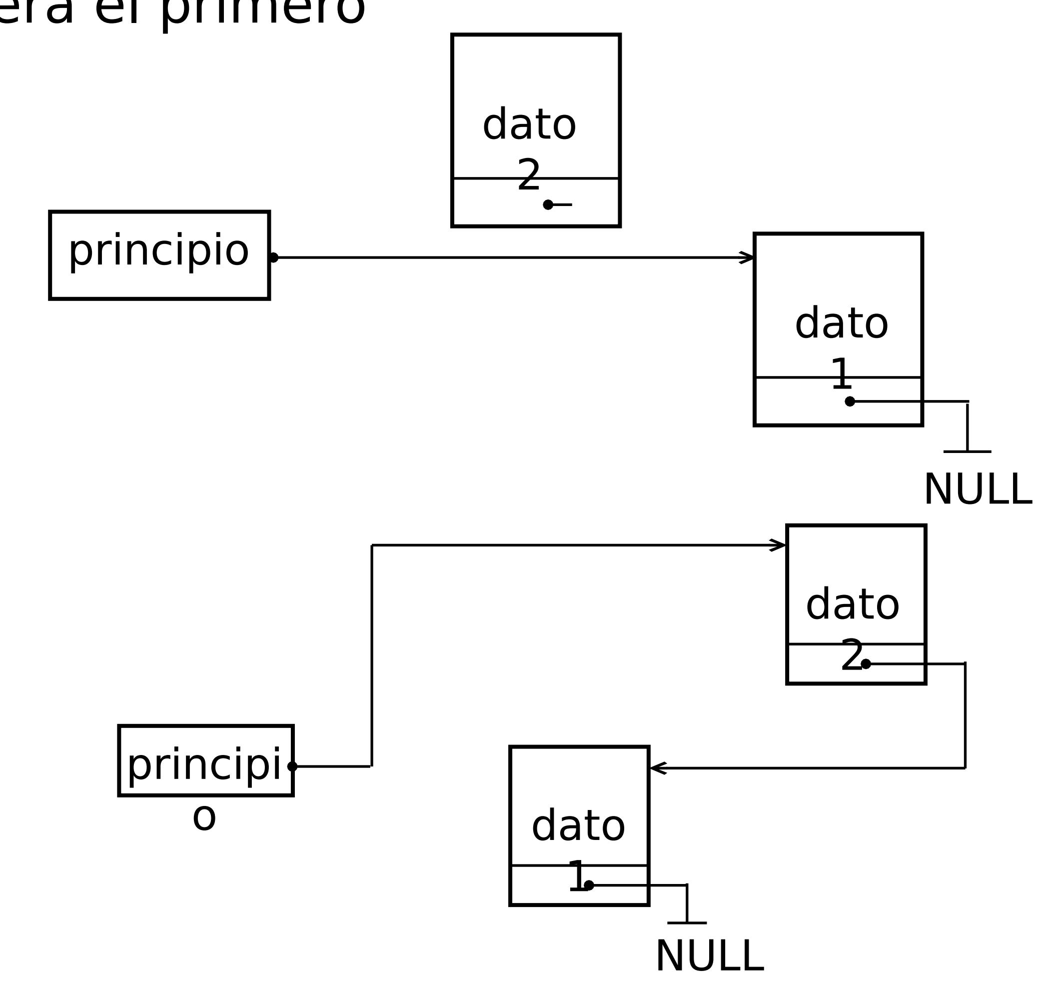
nsertar como último elemento

sertar como un elemento intermedio

Siempre que la lista este vacía el puntero de comienzo estará apuntando a NULL

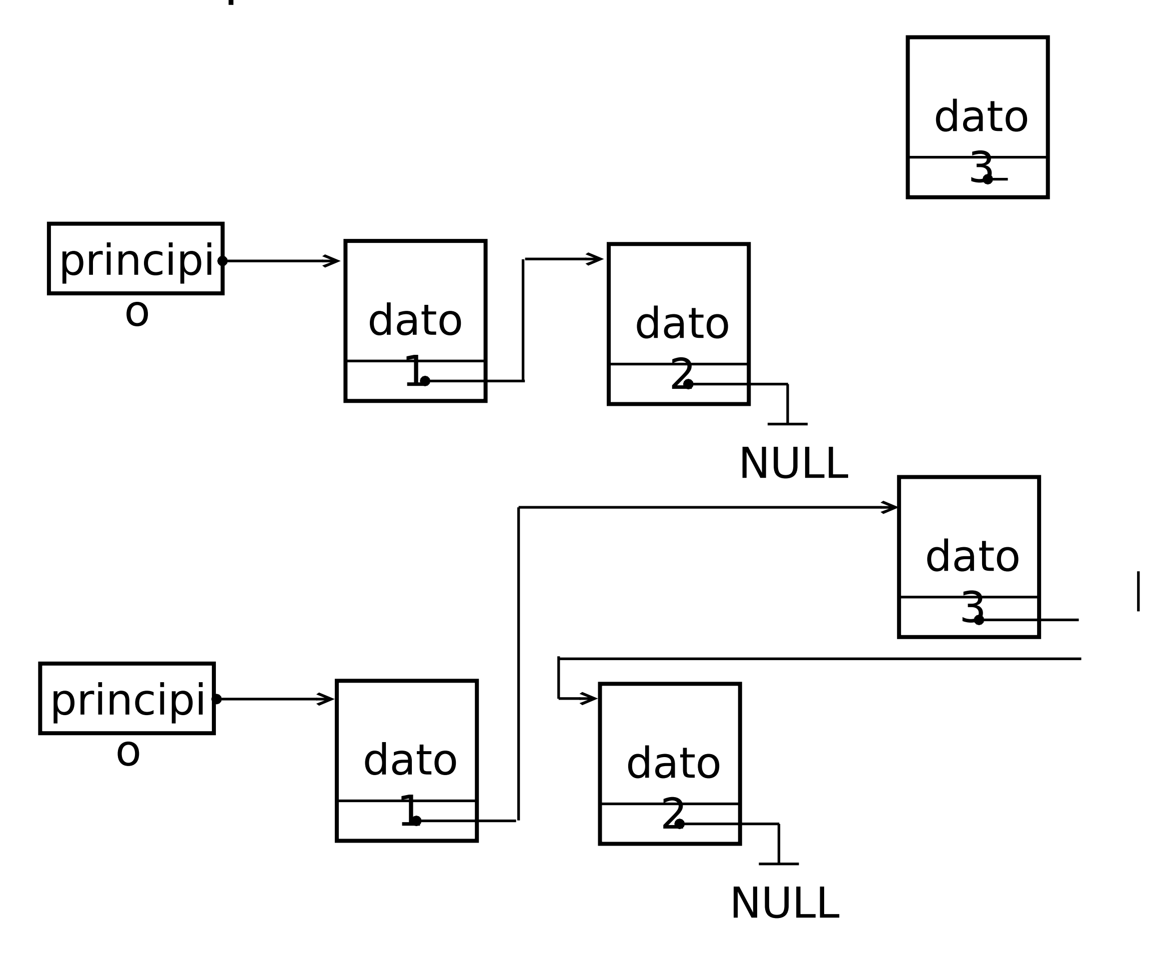
mo nuevo primer elemento

Cuando sucede que el nuevo elemento debe ser el primer elemento el indicador de principio deberá apuntar a este nuevo elemento y el indicador del elemento siguiente en el nuevo deberá apuntar al elemento que era el primero



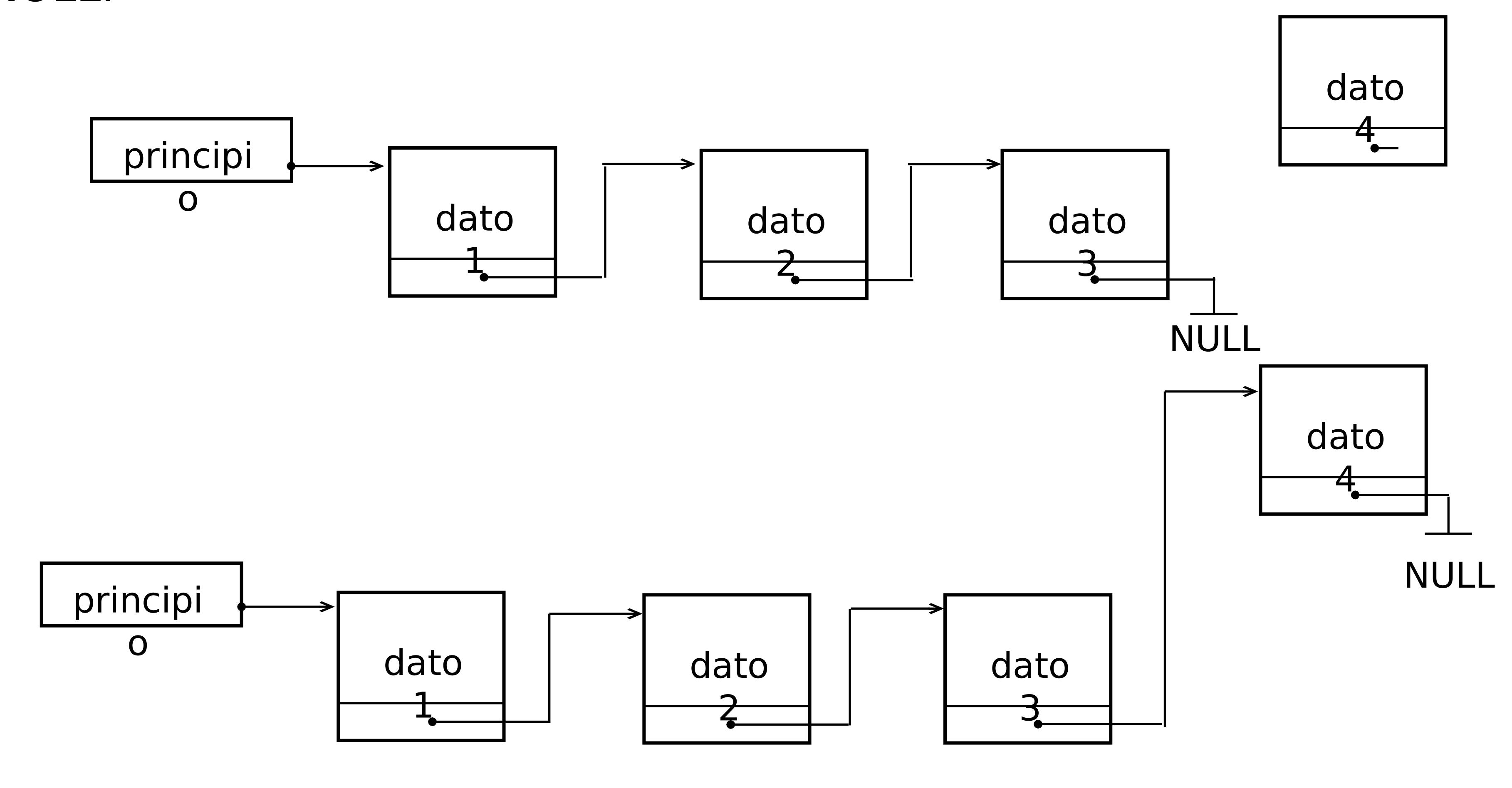
Como elemento intermedio

Esta ubicación es entre dos elementos existentes, entonces se debe cambiar el enlace del elemento anterior hacia el nuevo y el enlace del nuevo apuntará al posterior.



omo último elemento

El elemento a ser insertado debe ubicarse al final de la lista, por lo tanto el puntero siguiente del elemento final debe ahora apuntar al nuevo elemento y el puntero siguiente del nuevo debe apuntar a NULL.



Para leer los elementos de la lista se debe comenzar desde el indicador de comienzo y luego de presentar los datos, solo se debe actualizar el puntero con el valor del puntero siguiente de la estructura hasta llegar a encontrar el valor NULL.

Para buscar un determinado valor en la lista se debe hacer en forma secuencial por lo que hay que comenzar desde le valor que indica el puntero de comienzo hasta encontrar el valor deseado o hasta encontrar el NULL, que indicaría que no se ha encontrado el valor buscado

Para borrar un elemento encontrado en la lista pueden ocurrir tres casos:

Borrar el primer elemento

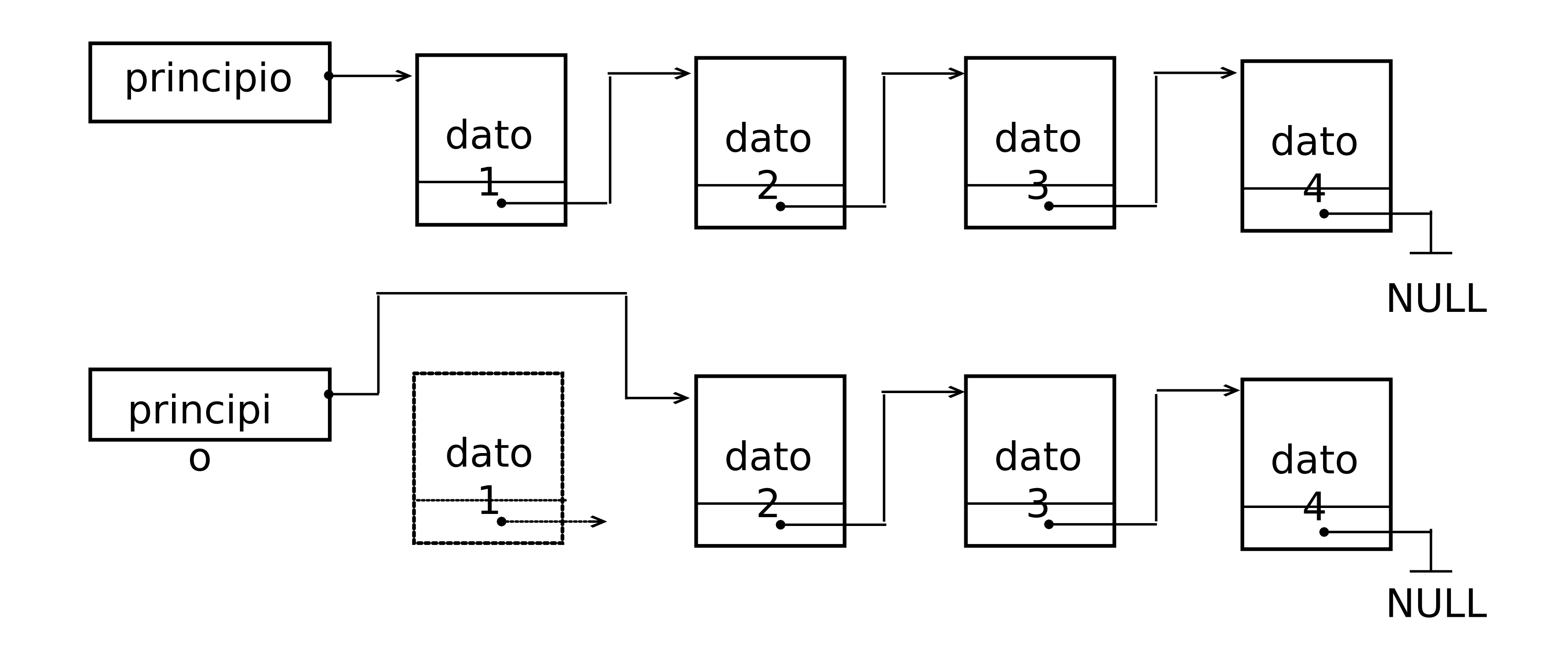
Borrar un elemento intermedio

Borrar el último elemento

rrar el primer elemento

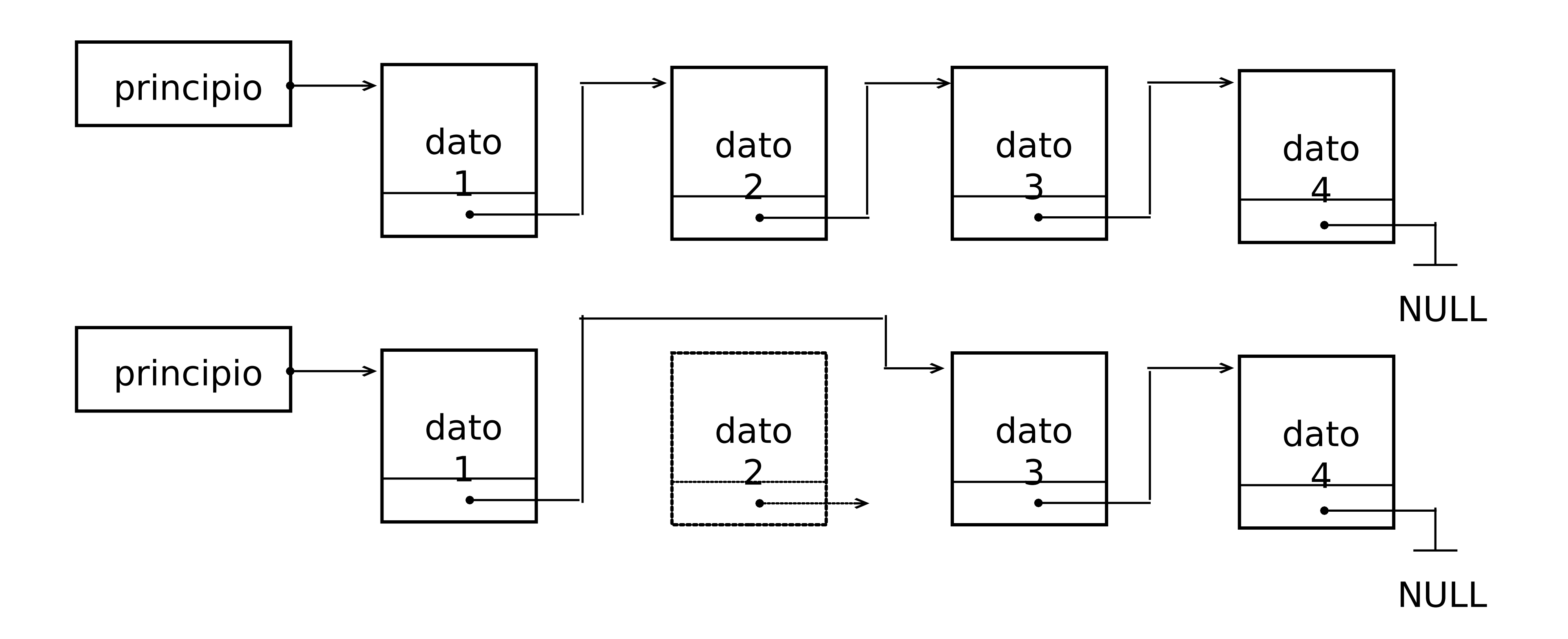
Se debe cambiar el puntero de comienzo a la dirección del segundo elemento

Luego se debe proceder a liberar la memoria del elemento borrado



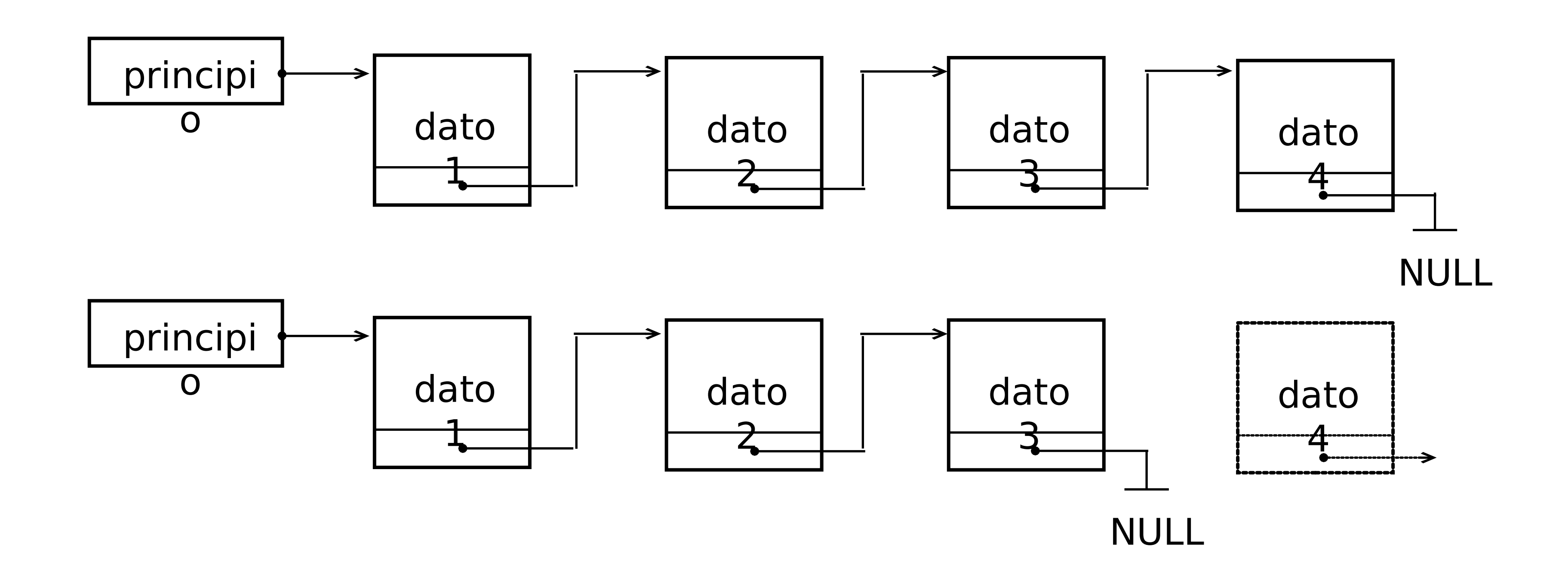
rrar un elemento intermedio

Para un elemento intermedio se debe hacer que el puntero siguiente del elemento anterior apunte al elemento siguiente al que será borrado y luego liberar la memoria de este



rrar el último elemento

El puntero siguiente del elemento anterior debe apuntar a NULL y luego liberar la memoria del elemento a borrar.



AGREGAR ELEMENTO

```
void ingresa(struct lis **p)
{
  struct lis *act,*aux;
  act=*p;
  if(!aux=(struct lis *)malloc(sizeof(struct lis)){
    printf("\n\nNo hay memoria\n");
    return;}
  endat(&aux);
  aux->sig=NULL;
  if(!*p){
    *p=aux;
    return;}
  while(act->sig)
    act=act->sig;
  act->sig=aux;
}
```

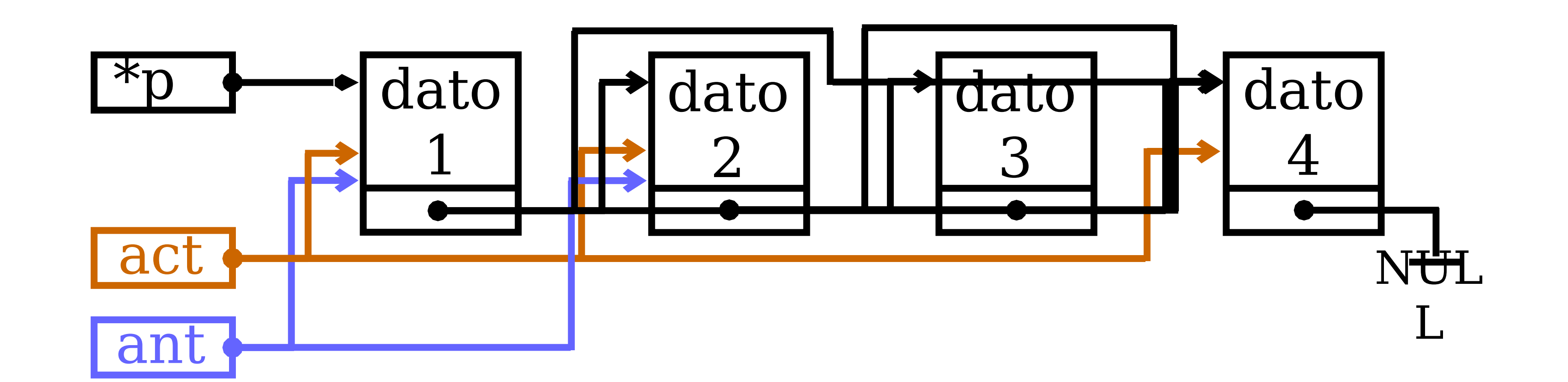
INSERTAR PRIMER ELEMENTO

```c
void ingresa(struct lis **p)
{
  struct lis *act,*aux,*ant;
  aux=(struct lis *)malloc(sizeof(struct lis));
  endat(&aux);
  if(!*p){
    *p=aux;
    aux->sig=NULL;
    return;}
  ant=act=*p;
  while((act->dato<aux->dato)&&act){
    ant=act;
    act=act->sig;}
  if(!act){
    ult_ele();}
  if(ant==act){
    *p=aux;
    aux->sig=ant;
    return;}
  ele_int();
}
```

*p
NUL
L
dato
1
dato
2
dato
3
NUL
L
act
ant

INSERTAR ELEMENTO INTERMEDIO

```c
void ingresa(struct lis **p)
{
  struct lis *act,*aux,*ant;
  aux=malloc(sizeof(struct lis)));
  endat(&aux);
  if(!*p)
    in_pr_el1();
  ant=act=*p;
  while((act->dato<aux->dato)&&act){
    ant=act;
    act=act->sig;}
  if(!act)
    in_ul_el();
  if(ant==act)
    in_pr_el();
  ant->sig=aux;
  aux->sig=act;
}
```



INSERTAR ULTIMO ELEMENTO

```
void ingresa(struct lis **p)
{
  struct lis *act,*aux,*ant;
  aux=malloc(sizeof(struct lis));
  endat(&aux);
  if(!*p)
    in_pr_el1();
  ant=act=*p;
while((act->dato<aux->dato)&&act){
  ant=act;
  act=act->sig;}
if(!act){
  ant->sig=aux;
  aux->sig=NULL;
  return;}
if(ant==act)
  in_pr_el();
el_in();
}
```

*p

dato 1

dato 2

dato 3

dato 4

act

ant

NULL

NULL

NULL

NULL

NULL

BORRAR PRIMER ELEMENTO

```
void borra(struct lis **p)
{
  struct lis *aux,*ant;
  int b;
  b=ingdat();
  aux=*p;
  ant=*p;
  while((b!=aux->dato)&&(aux)){
    ant=aux;
    aux=aux->sig;}
  if((*p==aux)&&(aux)){
    impdat(aux);
    if(condat()){
      *p=aux->sig;
      free(aux);
      return;}}
  if(aux){
    datint();
    return;}
  printf("\nDatos no existen");
}
```

*p

dato 1

dato 2

dato 3

dato 4

NUL L

aux

ant

BORRAR ELEMENTO INTERMEDIO

```c
void borra(struct lis **p)
{
  struct lis *aux,*ant;
  int b;
  b=ingdat();
  aux=*p;
  ant=*p;
  while((b!=aux->dato)&&(aux)){
    ant=aux;
    aux=aux->sig;}
  if((*p==aux)&&(aux)){
    boprida();
    return;}}
  if(aux){
    impdat(aux);
    if(condat()){
      ant->sig=aux->sig;
      free(aux);
      return;}}
  printf("\nDatos no existen");
}
```

BORRAR ULTIMO ELEMENTO

```c
void borra(struct lis **p)
{
  struct lis *aux,*ant;
  int b;
  b=ingdat();
  aux=*p;
  ant=*p;
  while((b!=aux->dato)&&(aux)){
    ant=aux;
    aux=aux->sig;}
  if((*p==aux)&&(aux)){
    bopriel
    return;}
  if(aux){
    impdat(aux);
    if(condat()){
      ant->sig=aux->sig;
      free(aux);
      return;}}
  printf("\nDatos no existen");
}
```

